Universal Cross-Assemblers

P.O. Box 384
Bedford, Nova Scotia
Canada
B4A 2X3

CROSS-16 META-ASSEMBLER
USER'S GUIDE

CROSS-16 USER'S MANUAL

version 1.16

Universal Cross-Assemblers
P.O. Box 384
Bedford, Nova Scotia
Canada B4A 2X3

Copyright (C) 1986 by Universal Cross-Assemblers.
All rights reserved.

LICENSE

Use of this product by any person other than the purchaser or on more than one computer at any specific time is strictly prohibited.

WARRANTY

Universal Cross-Assemblers will replace the Cross-16 source disk and/or reference manual once for a period of ninety days from the date of purchase if the original disk and/or reference manual are unreadable upon their return. The purchaser is urged to make and use a back-up copy of the source disk.

DISCLAIMER

Universal Cross-Assemblers, disclaims all liabilities to any damages or losses that may occur through the use of it's products or the breach of it's warranty, other than the single replacement of the original product.

ACKNOWLEDGEMENT

This program was made possible through the use of the Microsoft C Compiler Version 3.0 which is the property of:

Microsoft Corporation
10700 Northup Way
Bellevue, WA, USA
98004

To Heather

Table of Contents

1 Introduction..... 5

2 Universal Truths Concerning Cross-16..... 6

3 Running Cross-16 7

4 The Cross-16 Assembly Source File..... 9

4.1 Assembly Line Format..... 9

4.2 Labels..... 10

4.3 Operands..... 11

4.3.1 Numeric Constants..... 11

4.3.2 String Constants..... 12

4.3.3 Expressions..... 13

4.4 Cross-16 Assembler Directives..... 15

4.4.1 CPU - Table Declaration..... 16

4.4.2 DFB - Define Byte..... 17

4.4.3 DFL - Define Long Integer..... 18

4.4.4 DFS - Define Data Storage..... 19

4.4.5 DWL - Define Word Least..... 20

4.4.6 DWM - Define Word Most..... 21

4.4.7 END - End of Program..... 22

4.4.8 EQU - Equate..... 23

4.4.9 HOF - Hexadecimal Output Format..... 24

4.4.10 IF, ELSE, END - Conditional Assembly... 25

4.4.11 INCL - Include source file..... 27

4.4.12 ORG - Origin..... 28

4.4.13 PAGE - Page Control..... 29

4.4.14 SET - SET label to dynamic value..... 30

4.4.15 TITL - Title..... 31

5 The Cross-16 Listing File 32

5.1 Listing Format..... 32

5.2 Assembly Error Codes..... 32

6 The Cross-16 Hexadecimal File 34

7 The Cross-16 Instruction Table 35

7.1 Operand Types..... 36

7.1.1 The A Operand Type..... 38

7.1.2 The B Operand Type..... 39

7.1.3 The C Operand Type..... 40

7.1.4 The I Operand Type..... 41

7.1.5 The L Operand Type..... 42

7.1.6 The P Operand Type..... 43

7.1.7 The R Operand Type..... 44

7.1.8 The S Operand Type..... 45

7.1.9 The T Operand Type..... 46

7.2 Register Definition..... 47

7.3 Sorting the Instruction Table..... 50

8 Non-Assembly Error Messages..... 51

8.1 Warning Error Messages..... 52

8.2 Fatal Error Messages..... 54

9 Processor Manufacturers..... 56

1 Introduction

Congratulations! You have selected one of the most cost effective tools for developing assembly language programs for 8 and 16 bit microprocessors and microcontrollers. But first a few words on assembly language programing in general.

An assembly language program is a text source file where manufacturer's mnemonics or "memory aids" are used to directly represent the binary codes that a computer actually executes. An assembler is a computer program that converts these mnemonics into a their corresponding binary codes. The simple word "assembler" usually refers to a resident or self-assembler, which assembles programs for the same processor it runs on. ASM, supplied with the CP/M-80 operating system is an example of this. A cross-assembler, assembles programs for a single target processor which differs from it's host processor. Examples of these distributed by "the competition" may be found in the more technical computer and electronic journals.

Cross-16 from Universal Cross-Assemblers is a meta-assembler, in that it assembles programs for numerous different target processors. It reads an assembly language source file and a corresponding sorted assembler instruction table from disk, and writes an assembled listing and a hexadecimal machine language output file. By using a flexible instruction table structure, Cross-16 is designed to assemble machine language code for most microprocessors and microcontrollers with an address word of 24 or less bits. To further enhance it's flexibility, Cross-16 will produce a hexadecimal output file in the Intel and Motorola 8 and 16 bit formats.

As one proceeds throught this manual, one will pass from the broad generalities needed to use Cross-16, to the detailed specifics necessary to write a processor instruction table. If the user simply wishes to assemble code for one of the processors with an instruction table provided, then the user need not read section 7 on creating an instruction table, but should carefully study the notes provided in the example given for that processor. To successfully tailor a provided table or create a new one, the user must study the entire manual, and perhaps several of the provided tables.

2 Universal Truths Concerning Cross-16

The following are broad generalities for those already familiar with compilers and assemblers. Others need not fret, all of the following will be explained in greater detail in the appropriate sections of this manual.

Cross-16 is a two pass cross-assembler with an optional third pass if a phase error is detected.

All input and output files contain ASCII characters with each line terminated by an ASCII carrage-return and line-feed.

The most significant bit of all ASCII characters is reset by Cross-16 (i.e. ASCII_CHAR = ASCII_CHAR AND 7FH).

Only one processor assembly instruction or Cross-16 assembler directive is permitted per line.

All label declarations must be terminated with a colon ":", including labels for the EQU and SET directives.

All labels, expressions and operands are internally stored and manipulated using 32 bit signed integers.

Cross-16 makes no distinction between upper and lower case alphabetic characters.

Blank lines in the assembly source code are reproduced in the assembly listing but otherwise ignored.

Overflow errors, undefined labels, and invalid expressions are assigned a value of zero.

There are no restrictions on the character length of labels or processor instructions and all characters are significant. However, Cross-16 input lines must not exceed 128 characters in length.

Cross-16 does not support macros nor does it include a linker or librarian.

3 Running Cross-16

The enclosed disk contains the Cross-16 Meta-Assembler (C16.EXE), the Cross-16 Sort program (C16SORT.EXE) and sixteen processor tables (cpu.TBL) with sixteen corresponding example assembly source files (Tcpu.ASM). Universal Cross-Assemblers recommends that the user backs up the entire Cross-16 disk. Then copy C16.EXE and the table and example file for the processor that interests you on to a disk or directory that contains your favorite program editor. The following examples use the T8048.ASM and 8048.TBL files.

Cross-16 may then be run using the following command:

```
C16 sourcefile [-L listfile] [-H hexfile]
```

where the square brackets [] indicate optional items.

The "-L" tells Cross-16 to produce an assembly list file using the immediately following file name. The "-H" tells Cross-16 to produce a hexadecimal output file using the immediately following file name. If these are omitted the corresponding files will not be produced. All error codes will be displayed on the screen, regardless of whether a list file is created. The format of the hexfile is set using the HOF directive, explained in section 4.4.9 of this manual. The processor instruction table used by Cross-16 is also specified in the source file using the HOF directive as described in Section 4.4.1 of this manual. The order in which the source, listing and hex files are specified in the command line does not matter. Disk and directory names may be included in the file names.

Some examples are:

```
C16 T8048.ASM
                                    source:   T8048.ASM
                                    listing:  NONE
                                    hex:      NONE

C16 T8048.ASM -L B:T8048.LST
                                    source:   T8048.ASM
                                    listing:  B:T8048.LST
                                    hex:      NONE

C16 C:T8048.ASM -H T8048.HEX
                                    source:   C:T8048.ASM
                                    listing:  NONE
                                    hex:      T8048.HEX
```

```
C16 T8048.ASM -H T8048.HEX -L t8048.prn
                                        source:   T8048.ASM
                                        listing:  T8048.PRN
                                        hex:      T8048.HEX
```

Running Cross-16 produces the following on the screen:

```
A>C16 T8048.ASM

Cross-16 Meta-Assembler    PC/MS-DOS    Version 1.00
Copyright (C) 1986 Universal Cross-Assemblers

Starting Pass Number 1

Starting Pass Number 2

End of Assembly -- No Errors

A>
```

where "A>" is the operating system prompt. Since Cross-16 is a two pass meta-assembler, the source file is actually read twice. The processor instruction table is only read once and kept in RAM for the second pass. Any assembly errors will be displayed on the screen during the second pass, in addition to being placed in the listing if one was specified. The listing and hexadecimal files are written during the second pass only.

4 The Cross-16 Assembly Source File

The source file is the ASCII assembler source code to be assembled by Cross-16. From the source file, the processor instruction table is selected using the CPU directive, and the format of the hexadecimal file is set using the HOF directive.

The following sub-sections describe the fundamental building blocks which make up the assembly source file. Examples are given for each section, but these have been assembled and taken from the listing file; therefore the user does not provide the first 16 characters of these examples.

4.1 Assembly Line Format

Only one assembly instruction or assembler directive is permitted per line. The assembly line is free format, labels need not start in column 1 for example. Each line may contain some or all of the following sequence of identifiers:

```
line#     label:    operation      operand(s)     ; comment
```

Where...

Line# - is an optional decimal integer in the range of 0 to 65535 created by some editors representing the source code line number. If present, Cross-16 will ignore this field except to reproduce it in the listing.

Label - is a phrase starting with an alphabetic character or an underline "_" and ending with a colon ":", which is assigned the present value of the program counter or other user specified integer value. Characters within the label must be alphanumeric or an underline "_".

Operation - is a Cross-16 assembler directive, or processor assembly instruction defined in the instruction table. All operations must start with two alphabetic characters.

Operand(s) - are labels, constants or expressions representing integer values in the range of -2,147,483,648 to 2,147,483,647 and/or character strings. They may be embodied within an assembly instruction.

Comment - is a statement following a semi-colon ";" usually used to describe the assembly program. The ";" may be placed anywhere on the assembly line and Cross-16 ignores all characters following it on that line.

4.2 Labels

A label is an alphanumeric series of characters representing an integer in the range -2,147,483,648 to 2,147,483,647. The label field must start with an alphabetic ASCII character and must end with a colon ":", even when used with the EQU and SET assembler directives. After the first character, all remaining characters may be alphanumeric or an underline "_". Except for the EQU and SET directives, a label is optional and assigned the current value of the program counter. Labels may be of any length except that a Cross-16 input line cannot exceed 128 characters, and all characters are significant. Cross-16 makes no distinction between upper and lower case characters. A label may also stand alone on a line, in which case it will be assigned the current value of the program counter. A dollar sign "$" may be used as an operand representing the current value of the program counter.

The following examples are taken from an actual Cross-16 listing. They use the equate (EQU) directive described further in section 4.4.8. Very simply, the equate (EQU) directive assigns the label on its left, the value of the expression on its right.

```
0200                    ORG  0200H     ; Origin

                ;Valid examples of labels are:
                ;
1234 =          STRT1:  EQU  1234H     ;Label on directive
1234 =          LD_UP:  EQU  1234H     ;Label with "_"
0200            alone:                 ;Stand alone label
0200            DITO:

0987 =          LONG_LONG_LONG: EQU 987H ;Any length


                ;Invalid examples of labels are:
                ;
S0000           1LABEL: EQU  1234H     ;Starts with Number
S0000           MISS    EQU  1234H     ;Missing colon ":"
P1234 =         DUP:    EQU  1234H     ;Duplicate Label
P5678 =         DUP:    EQU  5678H     ;Duplicate Label
```

4.3 Operands

The following sub-sections describe numeric constants, string constants, and arithmetic and logical operators which in combination with labels may be used to form operands.

4.3.1 Numeric Constants

A numeric constant is an ASCII representation of a 32 bit signed integer in one of several number bases. All numeric constants must begin with a number, and their base is indicated by a trailing alphabetic character. The default base is base 10. The letter "Q" may also be used to specify octal numbers to avoid confusion between "O" and "0". The valid bases with their corresponding trailing alphabetic characters and character ranges are as follows:

B	Binary	Base 2	0-1
O	Octal	Base 8	0-7
Q	Octal	Base 8	0-7
D	Decimal	Base 10	0-9
H	Hexadecimal	Base 16	0-9, A-F

```
                ;Valid examples of numeric constants are:
                ;
00AF =          BIN1:     EQU  10101111B   ;Binary
00FF =          OCT1:     EQU  377O        ;Octal
003C =          OCT2:     EQU  74Q         ;Octal
FFFF =          DEC1:     EQU  -1          ;Decimal
000A =          DEC2:     EQU  10D         ;Decimal
00FF =          HEX1:     EQU  0FFH        ;Hexadecimal

                ;Invalid examples of numeric constants are:
                ;
V0000 =         HEX2:     EQU  0FFFFFFFFFH ;Too large
E007B =         DEC3:     EQU  123.4       ;Illegal decimal
U0000 =         HEX3:     EQU  FFH         ;No leading number
E0025 =         OCT3:     EQU  37 7Q       ;Illegal Blank
```

4.3.2 String Constants

String constants consist of a series of ASCII characters between two quotation marks ("). Cross-16 will convert these constants to a hexadecimal representation of their ASCII values. Lower case characters within string constants will be represented as such. A quotation mark (") cannot appear within a character string, for it will be interpreted as being the end of that string. If a quotation mark is needed, Universal Cross-Assemblers recommends that an apostrophe be used ('). The user may also terminate the string, insert the binary value of a quotation mark (22H), and then start another string. A string constant may also be used as an operand where applicable. In a DFB statement, a string constant may be of any length, bearing in mind that an assembly source line must not exceed 128 characters in length. When used as an operand, or in the DWM, DWL, DFL, SET or EQU directives, an error will be flagged if the string constant exceeds the length of the operand specified by the instruction. A string constant cannot be extended beyond it's present line without terminating the string with a quotation mark. Although only the first five or seven bytes of the string constant are shown in the listing, it is placed the hex file in its entirety.

The following examples use the define byte (DFB) directive as described in section 4.4.2. Very simply, the define byte (DFB) directive places the byte by byte value of the expressions on its right into memory, starting at the present memory location shown on the left.

```
0000                    ORG  0000H

                ;Valid examples of string constants are:
                ;
0000 5965612C20         DFB  "Yea, yea, yea!"
000E 41426162           DFB  "AB","ab"
0012 5468652064         DFB  "The dog Said 'Woof Woof!'"


                ;Invalid examples of string constants are:
                ;
T002B                   DFB  'Hello'   ;Using apostrophes
A002B 4D69737369        DFB  "Missing  ;Missing quote
```

4.3.3 Expressions

Cross-16 will accept numeric expressions made up of labels, constants and script brackets "{}" combined with logical and arithmetic operators. A list of recognized logical and arithmetic operators in there relative precedence follows where x and y represent integer values:

{ }	script parentheses
- y	Unary minus of y identical to 0-y or the twos complement of y
x * y	integer multiplication of x and y
x / y	integer division of x by y
x MOD y	integer remainder after division of x by y
x SHL y	Logical shift left of x by y bits and filled from the right with zeros
x SHR y	Logical shift right of x by y bits and filled from the left with zeros
x + y	integer addition of x and y
x - y	integer subtraction of y from x
NOT y	Ones complement of y
x AND y	Logical AND of x and y
x OR y	Logical OR of x and y
x XOR y	Exclusive OR of x and y
HIGH y	Binary value of high order byte only (y SHR 8)
LOW y	Binary value of low order byte only (y AND OFFH)
$	Present value of program counter

```
                ;Valid examples of operators are:
                ;
00FF =          EXPRES1:  EQU  2 OR 253       ;Logical OR
0003 =          EXPRES2:  EQU  "3" AND 15     ;Logical AND
FFFA =          EXPRES3:  EQU  NOT 0101B      ;Ones complement
0011 =          EXPRES4:  EQU  10B XOR CR     ;Exclusive OR
3400 =          EXPRES5:  EQU  0034H SHL 8    ;Logical Shift Left
0012 =          EXPRES6:  EQU  1234H SHR 8    ;Logical Shift Right
0078 =          EXPRES7:  EQU  LOW 5678H      ;Low Byte
0056 =          EXPRES8:  EQU  HIGH 5678H     ;High Byte
FFFF =          EXPRES9:  EQU  -(3 - 2)       ;Two's complement
4259 =          EXPRESa:  EQU  "AZ"+0FFH      ;Addition
0014 =          EXPRESb:  EQU  40-20D         ;Subtraction
4100 =          EXPRESc:  EQU  "A" * 256      ;Multiplication
0042 =          EXPRESd:  EQU  "BA"/256       ;Division
0001 =          EXPRESe:  EQU  10 MOD 3       ;Remainder

                ;Invalid examples of expressions are:
                ;
E0000 =         EXPRESf:  EQU  2**10D         ;Illegal operator
E0000 =         EXPRESg:  EQU  23*(-DEC2)     ;Wrong Parantheses
```

4.4 Cross-16 Assembler Directives

Cross-16 uses several common assembler directives combined with several of its own as listed below.

CPU	CPU table declaration
DFB	DeFine Byte
DFS	DeFine data Storage
DFL	DeFine Long integer
DWM	Define Word (High byte first)
DWL	Define Word (Low byte first)
END	END of program
EQU	EQUate label to specified value
HOF	Hexadecimal Output Format
IF, ELSE, ENDI	Conditional Assembly
INCL	INCLude source file
ORG	ORiGin
PAGE	PAGE control
SET	SET label to dynamic value
TITL	TITLe on listing

4.4.1 CPU - CPU Table Declaration

The CPU directive tells Cross-16 which processor instruction table is to be loaded during assembly.

The CPU directive has the following syntax:

```
label:    CPU  "cpu_file_name"   ;comment
```

Only the instruction table file name may be specified after with the CPU directive. A disk drive and/or directory may be included in the file specification. A label may be placed before the CPU directive, which will be assigned the current value of the program counter. The instruction table is only read once by Cross-16 during assembly, and all subsequent CPU directives will be ignored. A nonexisting CPU file name will result in a fatal error (see section 8.2). For further information on the format of the instruction table see section 7.

```
                ;Examples of the CPU directive
                ;
 0043                   CPU     "1802.TBL"    ; CPU TABLE
 0043                   CPU     "B:3870.TBL"  ; CPU TABLE


                ;Invalid examples of the CPU directive are:
                ;
S0043                   CPU     68000.TBL     ; Missing quotes
S0043                   CPU     8085          ; Cross-8 format
```

4.4.2 DFB - Define Byte

The DFB directive allows the user to define the value of storage areas on a byte by byte basis.

The DFB directive has the following syntax:

```
label:    DFB  expr1,expr2,...,expr(n)  ;comment
```

Except for a string constant, the result of each expression must represent an 8 bit value or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or a formula. There is no limit on the number of bytes that may be defined using a single DFB directive, except that the length of an Cross-16 input line must not exceed 128 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

```
 2000                    ORG  2000H

                 ;   Valid examples of the DFB directive are:
                 ;
 2000 5061746965NOTE:    DFB  "Patient Lead Off",CR,LF
                                        ;ASCII string
 2012 000102             DFB  0,1,2     ;Integers
 2015 1548               DFB  LOW{$}, 77Q + 9  ;Expressions
 2017 3F44               DFB  3FH, 44H  ;Hexadecimal


                 ;Invalid examples of the DFB directive are:
                 ;
V2019 00                 DFB  03FFH     ;Too Large (10 Bits)
V201A 00                 DFB  -129D     ;Too small
```

4.4.3 DFL - Define Long Integer

The DFL directive allows the user to define the value of storage areas on a long integer or long word basis (a long integer is 32 bits or four bytes). There is no limit on the number of long integers that may be defined using a single DFL directive, except that the length of a Cross-16 input line must not exceed 128 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

The directive syntax is as follows:

```
label:    DFL  expr1,expr2,...,expr(n)  ;comment
```

The value of each expression must be representable using a 32 bit signed integer or an assembly error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or a formula. Although ASCII string constants may be used, an error will be flagged if they exceed 4 characters. The DFL directive stores 32 bit signed integers from the most significant byte to the least significant byte.

```
3000                    ORG  3000H

                ;Valid examples of the DFL directive are:
                ;
3000 00000002   CONST1: DFL  2          ;Numeric constant
3004 00003000           DFL  CONST1     ;Label
3008 00000048           DFL  77Q + 9    ;Expression
300C 12345678           DFL  12345678H ;Hexadecimal


                ;An invalid example of the DFL directive is:
                ;
V3012 00000000          DFL  "ABCDE"      ;Too Long
```

4.4.4 DFS - Define Storage

The define storage (DFS) directive may be used to reserve a section of memory with unspecified contents during assembly.

The DFS directive has the following syntax:

```
label:    DFS  expression     ;comment
```

The expression can be of any form which represents a 32 bit positive integer value, but only one expression is allowed. The value of the expression is added to the program counter and assembly continues. Except for the changed value of the program counter, no values are written to the hex file.

```
5000                    ORG  5000H

                ;Valid examples of the DFS directive are:
                ;
5000            STOR_B:  DFS  20 * 1    ;Reserve 20 bytes
5014            STOR_W:  DFS  20 * 2    ;Reserve 20 words
503C            STOR_L:  DFS  20 * 4    ;Reserve 20 long


                ;Invalid examples of the DFS directive are:
                ;
V508C           STOR_E:  DFS  "ABCDE"   ;Too Long
V508C           STOR_F:  DFS  - 20      ;Negative Value
```

4.4.5 DWL - Define Word (Least Significant Byte First)

The DWL directive allows the user to define the value of storage areas on a word by word basis (one word is two bytes). There is no limit on the number of words that may be defined using a single DWL directive, except that the length of a Cross-16 input line must not exceed 128 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

The DWL directive has the following syntax:

```
label:    DWL  expr1,expr2,...,expr(n)  ;comment
```

The result of each expression must represent a 16 bit integer value or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or a formula. Although ASCII string constants may be used, an error will be flagged if they exceed 2 characters in length. The DWL directive will store the least significant byte of the 16 bit value before the most significant byte. By using either the DWM or DWL directives, words may be placed in memory in a format which corresponds to the format used by the target processor.

```
 4000                       ORG  4000H

                 ;Valid examples of DWL directive are:
                 ;
 4000 0000010002CONST2:     DWL  0,1,2      ;Numeric constants
 4006 06004800              DWL  LOW($),77Q+9  ;Expressions
 400A 3F004400              DWL  3FH,44H    ;Hexadecimal
 400E FFFF                  DWL  0FFFFH     ;up to 16 bits


                 ;Invalid examples of DWL directive are:
                 ;
V4010 0000                  DWL  "ABC"      ;Too Long
V4012 0000                  DWL  $ * 7777Q  ;Too Large
```

4.4.6 DWM - Define Word (Most Significant Byte First)

The DWM directive allows the user to define the value of storage areas on a word by word basis (one word is two bytes). There is no limit on the number of words that may be defined using a single DWL directive, except that the length of a Cross-16 input line must not exceed 128 characters. Although only the first 5 or 7 bytes of data will be shown in the listing, it is included in the hex file in its entirety.

The DWM directive has the following syntax:

```
label:    DWM  expr1,expr2,...,expr(n)  ;comment
```

The result of each expression must represent a 16 bit integer value or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or a formula. Although ASCII string constants may be used, an error will be flagged if they exceed 2 characters in length. The DWM directive will store the the most significant byte of the 16 bit value before the least significant byte. By using either the DWM or DWL directives, words may be placed in memory in a format that corresponds to the format used by the target processor.

```
 3000                      ORG  3000H

                ;Valid examples of the DWM directive are:
                ;
 3000 0000000100CONST1:    DWM  0,1,2      ;Numeric constants
 3006 3000                 DWM  CONST1     ;Label
 3008 00080048             DWM  LOW{$},77Q+9  ;Expressions
 300C 003F0044             DWM  3FH,44H    ;Hexadecimal
 3010 FFFF                 DWM  0FFFFH     ;up to 16 bIts

                ;Invalid examples of the DWM directive are:
V3012 0000                 DWM  "ABC"      ;Too Long
V3014 0000                 DWM  $ * 999D   ;Too Large
```

4.4.7 END - End of Program

The end of assembly (END) directive is optional, but when used it will be the last line of the assembly source file assembled (the remainder being ignored).

This directive has the following syntax:

```
label:    END  expression     ;comment
```

The expression is optional, but may represent any positive 16 or 24 bit integer value, depending on the hexadecimal output format in use. When an expression is given, its value will be included as the address in the final line of the hexdecimal machine language file, representing the starting address of the assembly program. If the END directive or expression are not included, this starting address will default to zero.

```
                ;Valid examples of the END directive are:
                ;
 0000                    END             ;Simple format
 0000           THE_END: END  RESET      ;With starting address
 0100                    END  0100H      ;Famous starting address


                ;An invalid example of the END directive is:
                ;
T0001                   END  1,2        ;Multiple operands
```

4.4.8 EQU - Equate

The equate (EQU) directive may be used to assign an integer value to a label.

The EQU directive has the following syntax:

```
label:    EQU  expression     ;comment
```

Neither the label or the expression are optional in this directive. The expression may consist of any numeric constant, character string or formula whose value can be represented in 32 bits. Cross-16 will place the value of expression in the label field followed by an equal sign "=" to show that this value is not the current value of the program counter. Defining a label more than once, or placing multiple expressions after the EQU directive, will result in an assembly error.

```
                ;Valid examples of the EQU directive are:
                ;
 0013 =         CR:       EQU  13H        ;ASCII Carrage Return
 000A =         LF:       EQU  10D        ;ASCII Line Feed
 5012 =         CNTR:     EQU  $          ;Program counter
 1300 =         EXPR2:    EQU  CR SHL 8   ;Formula


                ;Invalid examples of the EQU directive are:
                ;
T0021 =         TWICE:    EQU  33, -6     ;Multiple expressions
E0000 =         WHAT:     EQU             ;Missing expression
Y0D05 =                   EQU  3333       ;Missing Label
```

4.4.9 HOF - Hexadecimal Output Format

The hexadecimal output format (HOF) selects the format of the hexadecimal machine language output file.

The HOF directive has the following syntax:

```
label:    HOF  "format"    ;comment
```

A label may be included with the HOF directive, which will be assigned the current value of the program counter. The HOF directive is optional, and if not included Cross-16 will default to the "INT16" format. The HOF directive may appear anywhere in the assembly source file. If it appears more than once with different hexadecimal formats specified, the format of the hexadecimal file will change without an error code being generated.

The HOF directive partially controls the format of the assembled listing. If a HOF directive in not used, or one of the 16 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 32 bit value preceding the EQU and SET directives, 24 bit addresses, and up to 7 bytes of code on each line. If one of the 8 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 16 bit value preceding the EQU and SET directives, 16 bit addresses, and up to 5 bytes of code on each line. This allows the listing format to correspond to the address word of the target processor.

```
  6500                ORG   6500H

                ;Valid examples of the HOF directive are:
                ;
006500                    HOF  "INT16"  ;Intel 16 bit hex
 6500               HOF  "INT8"       ;Intel 8 bit hexal
006500                    HOF  "MOT16"  ;Motorola 16 bit hex
 6500               HOF  "MOT8"       ;Motorola 8 bit hex

                ;Invalid examples of the HOF directive are:
                ;
H6500                HOF  "BIN8"       ;Illegal hex format
U6500                HOF  INT          ;Cross-8 syntax
```

4.4.10 IF, ELSE, and ENDI - Conditional Assembly

Cross-16 supports conditional assembly using the IF, ELSE and ENDI directives to define areas of the source file which are or are not to be assembled. This feature is usually used to reconfigure a single assembly language program for different hardware environments.

Conditional assembly has the following syntax:

```
IF   expression     ;comment

  line 1
  line 2
  .
  .
  .
  line n

ELSE                ;comment

  line 1
  line 2
  .
  .
  .
  line n

ENDI                ;comment
```

Upon encountering an IF statement Cross-16 evaluates the single expression following it. All labels used in this expression must be defined previous to the IF. If the expression evaluates to zero, the statements between the IF and either an ELSE or an ENDI are not assembled, but are shown in the listing. If the expression results in a non-zero value, the statements between the IF and either an ELSE or an ENDI are assembled. The ELSE is an optional directive, allowing only one of the two sections of the source file within the IF block to be assembled. All conditional blocks must have an IF directive and an ENDI directive, the ELSE directive being optional. If blocks may be nested 16 deep before a fatal error occurs.

An example of conditional assembly follows, where a microcontroller writes to a printer, which is connected to either a RS-232C serial port or a Centronics parallel port, but not both.

```
6900                        ORG  6900

                ;An example of Conditional Assembly is:
                ;
0000 =          FALSE:      EQU  0D
FFFF =          TRUE:       EQU  NOT FALSE


                ;In this example use the RS-232C port
                ;
FFFF =          RS232C:     SET  TRUE
0000 =          CENTRONICS: SET  FALSE


                ;Conditional Block Starts Here
                ;
FFFF =                      IF   RS232C

0034 =          IO_PORT:    EQU  34H        ;RS-232C Port

                            ENDI


0000 =                      IF   CENTRONICS

                IO_PORT:    EQU  44H        ;Centronics Port

                            ENDI
```

4.4.11 INCL - Include Source File

The INCL directive tells Cross-16 to insert a specified source file into the one specified in the command line, or a previous include file.

The INCL directive has the following syntax:

```
label:    INCL  "source_file_name"   ;comment
```

Included source files may only be nested a maximum of 3 deep before a fatal error occurs. A disk drive and/or directory may be included in the file specification. A label may be placed before the INCL directive, which will be assigned the current value of the program counter. All included files are read once each pass, and are included in there entirety in the listing. Should an END directive be encountered in an included file, assembly of all source files will cease at that point. A nonexistant include file will result in a fatal error (see section 8.2).

```
                ;An example of a INCL directive is:
                ;
23A4                    CPU     "A:IO.ASM"
```

4.4.12 ORG - Origin

The origin (ORG) directive allows the user to specify the value of the program counter during assembly.

The ORG directive has the following syntax:

```
label:    ORG  expression     ;comment
```

The ORG directive may be used as often as desired, but Cross-16 will not flag areas that may be defined more than once in a single source file. The expression is not optional in this directive and may consist of any numeric constant, character string or formula whose value can be represented in a 16 or 24 bit positive integer, depending on which hexadecimal format has been declared using the HOF directive. Cross-16 will place the new value of the program counter in the address field. Placing no or multiple expressions after the ORG directive, will result in an error being flagged.

```
                ;Valid examples of the ORG directive are:
                ;
0000            RESET:    ORG  0          ;A common begining
0100                      ORG  0100H      ;A famous start


                ;Invalid examples of the ORG directive are:
                ;
T8000           TWICE2:   ORG  8000H,33   ;Multiple expressions
E0000           WHERE:    ORG             ;Missing expression
```

4.4.13 PAGE - Page Control

The page (PAGE) directive has two functions. In the following format:

```
label:    PAGE                ;comment
```

Cross-16 will insert a form feed into the listing before the next line of the listing. This causes the printer to continue the listing on a new page.

However, the next format:

```
label:    PAGE      expression        ;comment
```

will set the page length to the value of the expression. Each time the page length is reached, Cross-16 inserts a form feed into the listing. All positive integer values except 1 and 2 are legal. If the expression has a value of zero, or a pagesize is not specified, form feeds will not be inserted into the listing.

```
A000                    ORG    0A000H

                ;Valid examples of the PAGE directive are:
                ;
A000                    PAGE   56
A000                    PAGE    0  ;No form feeds
A000                    PAGE   60
A000                    PAGE       ;Form feed before next line
```

4.4.14 SET - Set

The set (SET) directive may be used to assign an integer value to a label. It is similar to the EQU directive except that the value of the label may be redefined using additional SET directives elsewhere in the assembly source file.

The SET directive has the following syntax:

```
label:    SET  expression     ;comment
```

Neither the label or the expression are optional in this directive. The expression may consist of numeric constant, character string or formula whose value can be represented as a 32 bit signed integer. Cross-16 places the value of expression in the address field followed by an equal sign "=" to show that this value is not the current value of the program counter. Placing no or multiple expressions after the SET directive, will result in an assembly error being flagged.

The SET directive is most commonly used for controlling conditional assembly (See section 4.4.5).

```
                ;Valid examples of the SET directive are:
                ;
 0000 =         CPM:      SET  0          ;CPM is false
 FFFF =         MS_DOS:   SET  - 1        ;MS_DOS is true


                ;Invalid examples of the SET directive are:
                ;
T0021 =         DOUBLE:   SET  33, -6     ;Multiple expressions
V0000 =         TOO_BIG:  SET  1FFFFFFFFH ;Value too large
```

4.4.15 TITL - Title of Listing

The title (TITL) directive will cause Cross-16 to place the character string following the directive at the top of each page of the listing.

The title directive has the following syntax:

```
label:    TITL   "character string"     ;comment
```

Both the label and the comment are optional for this directive. The "character string" is not, however, and must at the very minimum be a null string "". There is no limit to the length of the character string, except that a Cross-16 input line cannot exceed 128 characters. Cross-16 does not require TITL directive to produce a correctly formatted listing.

```
A800                    ORG  0A800H

                ;Valid examples of the TITL directive are:
                ;
A800                    TITL  "Cross-16 Test File"
A800                    TITL  "User's Choice"
```

5 The Cross-16 Listing File

If requested using the -L directive in the command line, Cross-16 will produce a listing file during the second pass of the assembly source file.

The HOF directive partially controls the format of the assembled listing. If a HOF directive in not used, or one of the 16 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 32 bit value preceding the EQU and SET directives, 24 bit addresses, and up to 7 bytes of code on each line. If one of the 8 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 16 bit value preceding the EQU and SET directives, 16 bit addresses, and up to 5 bytes of code on each line. This allows the listing format to correspond to the address word of the target processor.

5.1 Listing Format

The listing is the original assembly source file, with 16 or 24 additional ASCII characters inserted at the beginning of each line. Should an assembly error occur, an alphabetic error code will be placed in the first column of the listing, which is normally a " " blank. The next four or six characters represent the hexadecimal value of the program counter. Following the program counter, is another blank, followed by the hexadecimal value of the assembly instruction or assembler directive. This value will not be displayed after the fifth or seventh byte, but will be placed in the hexadecimal file in its entirety.

The EQU, SET and conditional assembly directives are exceptions to these rules, and further information on them may be found in section 4.4 of this manual.

5.2 Assembly Error Codes

When Cross-16 detects an error during assembly, a single character error code is placed in the first column of the assembly listing on the line in which the error occurred. The error code and assembled line are also shown on the video display, so that the user does not have to search the listing (or even generate it) to check for assembly errors. As Cross-16 searches the instruction table, it will display the first error code generated for the given assembly line, even though other errors may occur. Therefore, certain combinations of instructions and format errors can produce a misleading error code. If the code does not seem to be relavent, look for other possible errors. Errors are included thoughout the directive and instruction examples in this manual, and the alphabetic error codes represent the following:

A ASCII Error - Illegal format of an ASCII character string, usually a missing quotation mark (").

B Bracket Error - Unequal numbers of left and right script brackets in an expression "{}".

E Expression Error - Unable to evaluate expression as written.

H Hex Foramt Error - Illegal hexadecimal output format specified in HOF directive.

L Line Error - Unexpected characters found in assembly line, (missing ';').

P Phase Error - Value of label changes with successive assembler passes.

S Syntax Error - Cross-16 assembler directive or processor instruction cannot be identified.

T Too Many Error - Too many arguments are contained in the Cross-16 assembler directive or processor instruction.

U Undefined Label - Label used in expression or opcode is undefined. Often caused by hexadecimal constant which does not start with a number.

V Value Error - Operand defined in Cross-16 assembler directive or processor instruction is incompatable with given value.

Y Symbol Error - Missing or illegal symbol (Label).

6 The Cross-16 Hexadecimal File

Cross-16 supports the Intel and Motorola 8 and 16 bit hexadecimal formats. The Intel 16 bit format "INT16" will be produced by default, and any one of the four may be selected using the HOF directive (see section 4.4.9). Examples of each are provided below for the assembly source file SHOW.ASM.

```
                ; File SHOW.ASM
                ;
A:SHOW.HEX              HOF  "INT8"          ;Hex Output Format
0100                    ORG  0100H
0100 4865786164BEGIN:   DFB  "Hexadecimal!" ;Character String
010C                    END  BEGIN
```

INT8 -- Intel (8 bit format)

```
:0C0100004865786164656369 6D616C217D
:00010001FE
```

INT16 -- Intel (16 bit format)

```
:020000020000FC
:0C0100004865786164656369 6D616C217D
:00010001FE
```

MOT8 -- Motorola (8 bit format)

```
S10F0100486578616465636 96D616C2179
S9030100FB
```

MOT16 -- Motorola (16 bit format)

```
S2100001004865786164656369 6D616C2178
S9030100FB
```

The hexadecimal output file is controlled using the -H directive in the Cross-16 command line as shown in section 3.

7 The Cross-16 Instruction Table

The Cross-16 instruction table defines the instructions and registers of the individual microprocessors and microcontrollers. If one is simply going to use one of the tables provided, this section of the manual may be skipped, but the example assembly file for the target processor should be studied carefully.

There are numerous examples of instruction tables included with Cross-16 which may be studied to gain further insight into the process of creating one. In addition to using the provided tables, Cross-16 provides many other options. The simplest, is that the provided tables may be modfied to generate a cross-assembler that is more specific to a particluar processor, rather than one that is for a family of processors. This is usually just a matter of deleting or adding several instructions. Therefore, as manufacturers produce new processors based on older processor families, the user's development software will not become obsolete. Another feature is the ability to alter the manufacturers assembly mnemonics to a format which better suits the user's specific needs or preferences. A single set of common instruction mnemonics could be developed for example, and by only changing the opcodes in the instruction table, these same mnemonics could be assembled into machine language for a number of different processors. Only instructions that are extremely hardware specific would have to be re-examined. Using similar methods, Cross-16 can be used to ease software upgrading to a more recent processor, especially if both processors are from the same manufacturer, and have similar instruction sets. Undocumented instructions, that have been found to exist in several common microprocessors may also be added.

The instruction table consists of an alphabetically ordered listing of all the processor's instructions, the format and placement of it's operands, and the corresponding hexadecimal machine language or opcode. The instruction table must be sorted by the supplied program C16SORT.COM before use. Each instruction must be on a separate line and followed by an ASCII carrage return and line feed. An operand may be placed anywhere within the instruction, with its characteristics enclosed in a set of script brackets "{}". The ASCII hexadecimal opcode for the instruction is placed between the carat "^" and colon ":". From this field Cross-16 determines both the value and length of the instruction's opcode. There are no limits on the length of the instruction or it's opcode, except that a Cross-16 input line cannot exeed 128 characters. During assembly, the supplied opcode is converted to a binary value and logically ORed with any operands defined within the script brackets "{}". Following the last instruction an optional asterisk "*" may be used to signify the start of a section of equate (EQU) assembler directives, which are normally used to define register values. During assembly these are treated in an identical fashion to the an EQU directive in the assembly source file (see section 4.4.8). The syntax of the instruction table is as follows:

```
Instruction {operand(s)} ^ Hexstring :
     "          "       "      "    "
     "          "       "      "    "
     "          "       "      "    "
     "          "       "      "    "
     "          "       "      "    "
*
label:   EQU  expression
  "       "       "
  "       "       "
  "       "       "
```

7.1 Operand Types

The instructions and operands are combined into a single expression on the left side of the "^". There must not be blanks anywhere in this field. Any blanks that may be specified in the processor's assembly language programming manual must be excluded. This will not prevent the user from placing blanks or tabs in the assembly source file. Operands within an instruction are optional, and there is no limit to their number.

For example, in the 8085.TBL instruction table, the enable interrupt instruction, which does not have any operands, appears as follows:

```
EI^FB:
```

When assembled by Cross-16, a listing using this instruction appears as follows:

```
 6000                    ORG  6000H
                ;Valid instructions without operands are:
                ;
 6000 FB                 EI            ;Enable Interrupts
 6001 FB         Interr: ei            ;Case does not matter

                ;Invalid instructions without operands are:
                ;
L6002                    EI MODEM      ;Illegal Operand
S6002                    IE            ;Syntax Error
```

As stated previously, an operand is defined by placing it in script brackets in it's proper location within the instruction. The operand definition, within the brackets has the following syntax:

{ Starting-Location Operand-Type Operand-Length }

Where:

Starting-Location is the position of the first bit of the operand within the instructions opcode. This value must be specified as a positive decimal number without a base indicator (D). The bit positions are numbered from left to right (Most significant bit first) starting at 0 and preceding to a maximum of 511.

Operand-Type is a single character which represents any changes that may be made to the opcode or checks that may be performed before it is placed there. The operand-types used in Cross-16 follow.

Operand-Length is number of signifcant bits of the operand which will be placed in the opcode, beginning with the least significant bit. The operand length must be a positive decimal number without a base indicator ranging from 1 to a maximum of 32.

7.1.1 The A Operand Type

A The operand is ORed with the opcode as specified by the starting-location and operand-length. A range check is performed on the value of the operand during assembly, to insure that it can be represented in a signed twos complement format in the number of bits specified by the operand-length. If it cannot, a value error "V" is generated and Cross-16 will proceed further in the instruction table looking for an instruction which will handle the operand. If a matching instruction with a sufficiently long bit-length is found by Cross-16, the value error "V" is not displayed and this new instruction is used. As an example, examine three "Rotate Memory Byte Left Through Carry" instructions for the 6809 processor.

```
ROL{011A5},X^6900:
ROL{016A8},X^698800:
ROL{16A16},X^69890000:
```

In these instructions, an offset and the contents of the X-Register are added to determine the address of the byte to be rotated. Note that the bit-length of the offset {011A5} and the length of the opcode increases with each subsequent instruction. The first instruction will support an offset range of +- 15, the second +- 127, while the third supports an offset range of +- 32767. When an (ROL OFFSET,X) instruction is found in the source assembly file, Cross-16 will evaluate the value of the offset. If it can be represented via twos complement notation in 5 bits, the first instruction is assembled, else, if it can be represented in 8 bits the the second instruction is assembled, else, if it can be represented in 16 bits the third instruction is assembled, else, an error occurs. It is essential that a leading zero be placed in the starting-location of the instructions with the shorter opcodes to insure that they appear first after being sorted by Cross-16 Sort. This procedure will enable Cross-16 to automatically select the shortest opcode for the ROL OFFSET,X instruction.

```
7000                    ORG  7000H
                ;Instructions with absolute range checks are:
                ;
000C =          OFFSET1:  EQU 000CH     ;Small offset
0068 =          OFFSET2:  EQU 0068H     ;Medium offset
5529 =          OFFSET3:  EQU 5529H     ;Large offset

7000 690C               ROL OFFSET1,X ;Short opcode
7002 698868             ROL OFFSET2,X ;Longer opcode
7005 69895529           ROL OFFSET3,X ;Longest opcode
```

7.1.2 The B Operand Type

B The operand is ORed with the opcode as specified by the starting-location and operand-length. A range check is performed on the value of the operand during assembly, to insure that it can be fully represented in an unsigned binary format in the number of bits specified by the operand-length. If it cannot, a value error "V" is generated and Cross-16 will proceed further in the instruction table looking for an instruction which will handle the operand. If a matching instruction with a sufficiently long bit-length is found by Cross-16, the value error "V" is not displayed and this new instruction is used. As an example, examine two AND memory with accumulator instructions for the 6801 processor.

```
ANDA{08B8}^9400:
ANDA{8B16}^B40000:
```

The first instruction is of the direct addressing type, used with memory locations 0 to 255. The second instruction is of the extended addressing type and may be used with memory locations 0 to 65,535. When an (AND A) instruction is found in the source assembly file, Cross-16 will check the length of the instruction's operand. If the operand is 8 or less bits in length, the first instruction will be assembled, but if the operand is 9 to 16 bits in length then the second instruction is assembled. It is essential that a leading zero be placed in the starting-location of the direct addressed instruction to insure that it appears first after being sorted by C16SORT. This procedure will enable Cross-16 to automatically select the shortest opcode for the AND A instruction. It also enables many of the supplied tables to distinguish between register values and data locations.

```
 8000                    ORG  8000H
                ;Valid instructions with bit range checks are:
                ;
 0067 =         Page0:   EQU  0067H    ;Page zero
 1234 =         Not0:    EQU  1234H    ;Elsewhere
                ;
 8000 9467               AND A  PAGE0  ;AND accumulator
 8002 B41234             AND A  NOT0   ;AND accumulator

                ;Invalid instructions with bit range checks are:
                ;
08005 9400              ANDA   1FFFFH ;Too large
W8007 9400              ANDA   "ABC"  ;Too long
```

7.1.3 The C Operand Type

C The operand is ORed with the opcode as specified by the starting-location and operand-length. This operand-type is identical to the B type except that a range check is not performed. It is necessary to assemble instructions that only make use of less significant bits of the operand. An example of this is the short branch (BR) instruction used by the 1802 series of microprocessors.

BR{8C8}^3000:

The short branch instruction loads the second byte of the instruction into the program counter, effectively creating a branch within the current 256 byte page. The value of the operand specified for the branch instruction in the assembly source file may indeed be longer than 8 bits, which would have been incorrectly flagged as an error if the "B" operand-type was specified.

```
 9000                    ORG  9000H
                ;A valid instruction without bit range checks is:
                ;
 9000 3057               BR   9057H    ;No Error Generated

                ;An invalid instruction without bit range  checks
is:
                ;
09002 3000              BR  1FFFFH    ;Too Long
```

7.1.5 The I Operand Type

I The high and low bytes are exchanged in a similar fashion to the DWL directive, and the result is ORed with the opcode as specified by the starting-location and operand-length. Since this operand type is used primarily with instructions that utilize the full sixteen bits of the operand, a range check is not performed, and Cross-16 sets bits 16 to 31 to zero.

```
JP{8I16}^C30000:
```

The Z80 is one processor that uses sixteen bit addresses with the least significant byte placed first in the opcode, as can been seen in the jump to address following (JP) instruction.

```
 B000                    ORG 0B000H
                 ;Valid instructions with exchanged bytes are:
                 ;
 B000 C32211             JP    1122H      ;Notice hex code
 B003 C34433             JP    3344H      ;Notice hex code

                 ;Invalid instructions with exchanged bytes are:
                 ;
0B006 C30000             JP   1FFFFH      ;Too Large
WB009 C30000             JP    "ABC"      ;Too Long
```

7.1.6 The L Operand Type

L A relative operand is calculated by Cross-16 during assembly and its high and low bytes are exchanged in the same manner as the DWL directive and the I operand type. The result is ORed with the opcode as specified by the starting-location and operand-length. This relative value is the value of the operand specified in the assembly language source file, minus the memory address of the byte immediately following the opcode. A range check is performed during assembly, to ensure that the calculated operand can be fully represented as a signed twos complement number in the number of bits specified in the operand-length. If this condition is not true, a range error "R" will be flagged, and Cross-16 will look for a matching instruction with a longer operand-length.

LCALL{8L16}^CD0000:

This type of inverted relative addressing is used by the 8096 family of microcontrollers for their long call subroutine instruction.

```
 C000                    ORG 0C000H
                 ;Valid relative exchanged instructions are:
                 ;
 C000 EF85D9             LCALL 9988H      ;Notice hex code
 C003 EF8AB8             LCALL 7890H      ;Notice hex code

                 ;Invalid relative exchanged instructions are:
                 ;
0C006 EF0000             LCALL 1FFFFH     ;Too Large
WC009 EF0000             LCALL "ABC"      ;Too Long
```

7.1.6 The P Operand Type

P The page operand-type is used primarily by the 8048 and 8051 family of microcontrollers for their 11 bit call and jump instructions. The page operand is similar to the "C" operand type except that bits 10 through 8 of the operand specified in the assembly language source file are ORed with bits 0 through 2 of the opcode, respectively. The operand is also ORed with the opcode as specified by the starting-location and operand-length. Like the "C" operand-type a range check is not performed.

ACALL{8P8}^1100:

An example of the page operand-type is the absolute call instruction of the 8051 family of microcontrollers.

```
D000                    ORG 0D000H
                ;An instruction with page addressing is:
                ;
D000 3123               ACALL 0D123H ;Absoulute call
```

7.1.8 The R Operand Type

R A relative operand is calculated by Cross-16 during assembly, and the result ORed with the opcode as specified by the starting-location and operand-length. This relative value is the value of the operand specified in the assembly language source file, minus the address of the byte immediately following the opcode. A range check is performed during assembly, to ensure that value of the calculated operand can be fully represented in a signed twos complement format in the number of bits specified in the operand-length. If this cannot be done, a value error "V" will be flagged, and Cross-16 will look for a matching instruction with a longer operand-length. This procedure enables Cross-16 to automatically select the shortest opcode for an instruction. Many processors use different mnemonics for the relative and absolute branching instruction. The user may wish to make these different mnemonics similar so that Cross-16 will select the shortest mnemonic.

BCC{8R8}^2400:

An example of an instruction that uses relative addressing, is the branch on carry clear instruction used by the 6805 micrcontroller.

```
 E000                    ORG 0E000H
                ;A valid instruction with relative addressing is:
                ;
 E000 2400              BCC  SUB1      ;Branch on carry clear
 E002           SUB1:   DFS  260D      ;Subroutine 1
 E106           SUB2:                  ;Subroutine 2

                ;An invalid instruction is:
                ;
RE106                   BCC  SUB1      ;Too far
```

7.1.9 The S Operand Type

S This operand-type is identical to the "R" type except than the relative operand is calculated as the operand value specified in the assembly source file minus the address of the last byte of the instruction. The relative operand is calculated by Cross-16 during assembly, and the result ORed with the opcode as specified by the starting-location and operand-length. A range check is performed to ensure that value of the calculated operand can be fully represented in a signed twos complement format in the number of bits specified by the operand-length. If this cannot be done, a value error "V" will be flagged, and Cross-16 will look for a matching instruction with a longer operand-length. This procedure enables Cross-16 to automatically select the shortest opcode for an instruction. Many processors use different mnemonics for the relative and absolute branching instruction. The user may wish to make these different mnemonics similar so that Cross-16 will select the shortest mnemonic.

```
BNO{8S8}^9800:
```

An example of an instruction that uses this operand-type is the "Branch If No Overflow" instruction used by the 3870 family.

```
 F000                    ORG 0F000H
                ;A valid example of alternative branching is:
                ;
 F000 9821              BNO  0F022H   ;Branch if no overflow

                ;An invalid example of alternative branching is:
                ;
0F002 9801              BNO  01000H    ;Too far
```

7.1.10 The T Operand Type

T This operand-type is identical to the "R" type except than the relative operand is calculated as the operand value specified in the assembly source file minus the address of the second to last byte of the instruction. The relative operand is calculated by Cross-16 during assembly, and the result ORed with the opcode as specified by the starting-location and operand-length. A range check is performed to ensure that value of the calculated operand can be fully represented in a signed twos complement format in the number of bits specified by the operand-length. If this cannot be done, a value error "V" will be flagged, and Cross-16 will look for a matching instruction with a longer operand-length. This procedure enables Cross-16 to automatically select the shortest opcode for an instruction. Many processors use different mnemonics for the relative and absolute branching instruction. The user may wish to make these different mnemonics similar so that Cross-16 will select the shortest mnemonic.

BCC{16T16}^64000000:

An example of an instruction that uses this operand-type is the "Branch If Carry Clear" instruction used by the 68000 family.

```
00F800                          ORG  0F800H

                        ;An example of "T" type branching is:
                        ;
00F800 64001002                 BCC  100804H    ;Branch if CC
```

7.2 Register Definition

An asterisk "*" may be placed on the line following the last assembly instruction in the instruction table. On subsequent lines, standard equate (EQU) directives may be used to define register values or whatever other labels the user may wish. For further information on the EQU directive see section 4.4.8. This definition section is optional, and may be placed in the assembly language source file instead.

As an example, the 8085 instruction table file (8085.TBL) showing the asterisk "*" and register definitions after the instruction table, is shown on the next two pages.

For further examples of instruction tables examine the cpu.TBL files on the Cross-16 source disk.

```
ACI{8B8}^CE00:
ADC{5B3}^88:
ADD{5B3}^80:
ADI{8B8}^C600:
ANA{5B3}^A0:
ANI{8B8}^E600:
CALL{8I16}^CD0000:
CC{8I16}^DC0000:
CMA^2F:
CMC^3F:
CMP{5B3}^B8:
CM{8I16}^FC0000:
CNC{8I16}^D40000:
CNZ{8I16}^C40000:
CPE{8I16}^EC0000:
CPI{8B8}^FE00:
CPO{8I16}^E40000:
CP{8I16}^F40000:
CZ{8I16}^CC0000:
DAA^27:
DADB^09:
DADD^19:
DADH^29:
DADSP^39:
DCR{2B3}^05:
DCXB^0B:
DCXD^1B:
DCXH^2B:
DCXSP^3B:
DI^F3:
EI^FB:
HLT^76:
INR{2B3}^04:
INXB^03:
INXD^13:
INXH^23:
INXSP^33:
IN{8B8}^DB00:
JC{8I16}^DA0000:
JMP{8I16}^C30000:
JM{8I16}^FA0000:
JNC{8I16}^D20000:
JNZ{8I16}^C20000:
JPE{8I16}^EA0000:
JPO{8I16}^E20000:
JP{8I16}^F20000:
JZ{8I16}^CA0000:
LDAXB^0A:
LDAXD^1A:
LDA{8I16}^3A0000:
LHLD{8I16}^2A0000:
LXIB,{8I16}^010000:
LXID,{8I16}^110000:
LXIH,{8I16}^210000:
LXISP,{8I16}^310000:
```

```
MOV{2B3},{5B3}^40:
MVI{2B3},{8B8}^0600:
NOP^00:
ORA{5B3}^B0:
ORI{8B8}^F600:
OUT{8B8}^D300:
PCHL^E9:
POPB^C1:
POPD^D1:
POPH^E1:
POPPSW^F1:
PUSHB^C5:
PUSHD^D5:
PUSHH^E5:
PUSHPSW^F5:
RAL^17:
RAR^1F:
RC^D8:
RET^C9:
RIM^20:
RLC^07:
RM^F8:
RNC^D0:
RNZ^C0:
RP^F0:
RPE^E8:
RPO^E0:
RRC^0F:
RST{2B3}^C7:
RZ^C8:
SBB{5B3}^98:
SBI{8B8}^DE00:
SHLD{8I16}^220000:
SIM^30:
SPHL^F9:
STAXB^02:
STAXD^12:
STA{8I16}^320000:
STC^37:
SUB{5B3}^90:
SUI{8B8}^D600:
XCHG^EB:
XRA{5B3}^A8:
XRI{8B8}^EE00:
XTHL^E3:
*
B: EQU 0
C: EQU 1
D: EQU 2
E: EQU 3
H: EQU 4
L: EQU 5
M: EQU 6
A: EQU 7
```

7.3 Sorting the Instruction Table

After being created or modified, the a Cross-16 instruction table must be sorted using C16SORT.EXE. This program searches the instruction table for errors, and finding none will sort the instructions alphabetically. The error messages are detailed and include the instruction at fault, and are therefore self explanatory.

C16SORT may be run using the following command:

C16SORT table_name

Where "table_name" is the file name of the instruction table to be sorted.

Therefore, the command

C16SORT 8096.TBL

will cause the C16SORT.EXE program to load and sort the 8096.TBL instruction table, and write a successfully sorted 8096.TBL instruction back on the disk drive.

8 Non-Assembly Error Messages

During the assembly of a source file, Cross-16 may generate a "Fatal Error" or "Warning" message. Usually these messages will be self-expanatory. Warning messages will not stop assembly, and are a result of a syntax error in the command line or a phase error. Encountering a fatal error will cause Cross-16 to terminate execution. When this occurs, all open files are closed, a fatal error message is sent to the screen and Cross-16 exits to the operating system.

The following example illustrates a warning message when an illegal option "-O" is placed in the Cross-16 command line. The error count "No Errors" refers to assembly errors only.

```
A>C16 T8085.ASM -O T8085.HEX

Cross-16 Meta-Assembler    PC/MS-DOS    Version 1.00
Copyright (C) 1986 Universal Cross-Assemblers

Warning -- Illegal Option Ignored
Warning -- Extra Source File Ignored

Starting pass number 1

Starting pass number 2

End of Assembly -- No Errors

A>
```

The following example illustrates a fatal error when a source file is placed in the Cross-16 command line which does not exist.

```
A>C16 NOFILE.ASM

Cross-16 Meta-Assembler    PC/MS-DOS    Version 1.00
Copyright (C) 1986 Universal Cross-Assemblers

Fatal Error -- Source File Did Not Open

A>
```

8.1 Warning Messages

Warning messages will not stop assembly, and are a result of a syntax error in the command line or a phase error. When such a problem occurs, "Warning -- " will appear on the screen, followed by the appropriate message. The message will be one of the following:

Illegal Option Ignored

Cross-16 only supports the -L and -H command line options. All others will be ignored, as will any file name that follows them.

-H Option Ignored -- Missing File Name

A hexadecimal output file has been requested, but a file name has not been provided. Assembly continues, but a hex file is not produced.

-L Option Ignored -- Missing File Name

A listing has been requested, but a file name has not been provided. Assembly continues, but a listing is not produced.

Phase Error

One or more labels have been assigned a value in the second pass that was not equal to the assigned value in the first pass. Cross-16 will perform a third pass to correct this problem.

Some processors have instructions with identical syntaxes but different operand sizes. Cross-16 assigns undefined labels a value of zero. This can cause Cross-16 to use the instruction with the shortest opcode in the first pass, and an instruction with a longer opcode in the second pass. This results in the addresses of the following labels being shifted. The 6805 for example has two unindexed jump instructions, the first only supports jumps to page zero (00H to 0FFH) while the second supports jumps to the entire address space. A jump to an

```
JMP{08B8}^BC00:
JMP{8B16}^CC0000:
```

undefined address during the first pass will use the two byte opcode, while the same jump to a now defined address, which is not on page zero, will use the three byte opcode during the second pass. Assembly in this manner will produce the shortest, fastest hexcode possible, but does require three passes. Users which are not pressed for either space or speed may wish to eliminate this third pass to reduce Cross-16's assembly time. This may be accomplished by:

1) Not using forward referenced labels

2) or removing the the shorter opcodes from the instruction table. The final version of your software can always be reassembled with the shorter opcodes installed.

Extra Source File Ignored

Only one source file can be specified in the command line. Additional files can be included via the INCL directive within the source file. This warning often follows the "Illegal Option Ignored" message.

Extra Hex File Ignored

Only one hexadecimal output file can be specified after the -H option in the command line. This warning often follows the "Illegal Option Ignored" message.

Extra List File Ignored

Only one listing can be specified after the -L option in the command line. This warning often follows the "Illegal Option Ignored" message.

8.2 Fatal Errors

Encountering a fatal error will cause Cross-16 to terminate execution. When this occurs, all open files are closed, "Fatal Error -- " followed by an appropriate message appears on the screen and Cross-16 exits to the operating system. The message will be one of the following:

No Source File Specified

A source file name was not included in the command line.

Source File Did Not Open

The source file included in the command line did not open. The file name is incorrect or on a disk drive or directory other than the one specified.

List File Did Not Open

The specified list file name is on a nonexistant disk drive or directory, or the directory is full.

Hex File Did Not Open

The specified hex file name is on a nonexistant disk drive or directory, or the directory is full.

Instruction Table Did Not Open

The instruction table file name specified after the CPU directive is incorrect or on a disk drive or directory other than the one specified.

Source File Read Error

Disk read error. Try again using a backup copy of your source file.

Disk or Directory Full

Delete unnecessary files and try again.

Too Many Include Files

Include files using the INCL directive may only be nested 3 deep.

Too Many Conditional Blocks

Conditional blocks using the IF, ELSE and ENDI directives may only be nested 16 deep.

Too Many Instructions

Cross-16 has run out of memory while loading the instruction table. This will only occur using a table for a 16 bit processor on a PC/MS-DOS computer with less than 192kB of RAM if the CPU directive is located a the begining of the source file.

Too Many Labels

Cross-16 has run out of memory when attempting to store a label in RAM. Reduce the number or the length of the labels in the source file or add more memory. A PC/MS-DOS computer with 640kB should store a minimum of 19,000 labels.

9 Processor Manufacturers

The following is an alphabetical list of the manufacturers from whom further information may be obtained regarding the microprocessors and microcontrollers for which instruction tables are supplied by Universal Cross-Assemblers:

Hitachi America, Ltd. Semiconductor and IC Sales 2210 O'Toole Avenue San Jose, CA 95131	64180
Intel Corp 3065 Bowers Avenue Santa Clara, CA 95051	8048 8051 8085 8086 8096
Mostek Corp 1215 W Crosby Road Carrollton, TX 75006	3870
Motorola Integrated Circuits 3501 Ed Bluestein Blvd Austin, TX 78721	6801 6805 6809 6811 68000
National Semiconductor Corp 2900 Semiconductor Drive Santa Clara, CA 95051	COP400
RCA Solid State Division Rte 202 Somerville, NJ 08876	1802
Rockwell International Microelectronic Devices Div 4311 Jamboree Rd Newport Beach, CA 92660	6502
Zilog Inc 1315 Dell Ave Cambell, CA 95008	Z8 Z80